Efficient coding of homogeneous textures using stochastic vector quantisation and linear prediction

D.Gimeno Gost and L.Torres

Abstract: Vector quantisation (VQ) has been extensively used as an effective image coding technique. One of the most important steps in the whole process is the design of the codebook. The codebook is generally designed using the LBG algorithm which uses a large training set of empirical data that is statistically representative of the images to be encoded. The LBG algorithm, although quite effective for practical applications, is computationally very expensive and the resulting codebook has to be recalculated each time the type of image to be encoded changes. Stochastic vector quantisation (SVQ) provides an alternative way for the generation of the codebook. In SVQ, a model for the image is computed first, and then the codewords are generated according to this model and not according to some specific training sequence. The SVQ approach presents good coding performance for moderate compression ratios and different type of images. On the other hand, in the context of synthetic and natural hybrid coding (SNHC), there is always need for techniques which may provide very high compression and high quality for homogeneous textures. A new stochastic vector quantisation approach using linear prediction which is able to provide very high compression ratios with graceful degradation for homogeneous textures is presented. Owing to the specific construction of the method, there is no block effect in the synthetised image. Results, implementation details, generation of the bit stream and comparisons with the verification model of MPEG-4 are presented which prove the validity of the approach. The technique has been proposed as a still image coding technique in the SNHC standardisation group of MPEG.

1 Introduction

In many synthetic and natural hybrid coding applications there is a need for good-quality homogeneous natural textures. Typical examples are model-based applications in which natural texture is mapped on a parametric model of a human face thus giving a natural reproduction. In a different type of application, such as distant virtual reality environments, the user may wish to download different types of textures in order to map them on different objects. Alternatively, the textures may be stored in the user's terminal to perform the mapping. Computer graphics overlays may also benefit of mapping different textures. In this context it is very useful to have a parameterised version of the texture of interest in order to achieve high compression, fast transmission and efficient downloading.

Vector quantisation (VQ) has been used extensively for many years as an effective image coding technique [1, 2]. One of the most important steps in the whole process is the design of the codebook. The codebook is generally designed using the LBG algorithm, which uses a large training set of empirical data that is statistically representative of the images to be encoded [3]. The LBG algorithm, although quite effective for practical applications, is computationally very expensive, and the resulting codebook has to be recalculated each time the type of image to be encoded changes. Stochastic vector quantisation (SVQ) provides an alternative means of generating of the codebook [4]. In the SVQ approach the codewords are generated by stochastic techniques instead of being generated by a training set representative of the expected input image. This means that the codebook is generated using a random number generator. In the original SVQ approach, white–Gaussian noise images of the same size as the subimages to be encoded, are passed through some shaping filter $H(z_1, z_2)$ whose output follows the selected model. Good results are achieved for moderate compression ratios. The objective of this paper is to present a modified and very improved version of the original SVQ technique, which combines 2-D stochastic vector quantisation (SVQ) and linear prediction (LP) to encode texture images using very high compression ratios with smooth quality degradation. Preliminary results of the approach were presented in [5]. Other vector quantisation works using synthetic codebooks have also been presented in [6] and [7].

2 Linear prediction in 2D

Given a greyscale digital image $u[x, y]$, a linear predictor can be defined by [2]:

$$\hat{u}[x, y] = \sum_k a_k u[x - \alpha_k, y - \beta_k] \quad (1)$$

IEE Proceedings online no. 19990332

DOI: 10.1049/ip-vis:19990332

Paper received 13th August 1998

D. Gimeno Gost is with DGG Software, Barcelona, Spain

L. Torres is with the Department of Signal Theory and Communications, Polytechnic University of Catalonia, Jordi Girona 1-3 08034, Barcelona, Spain

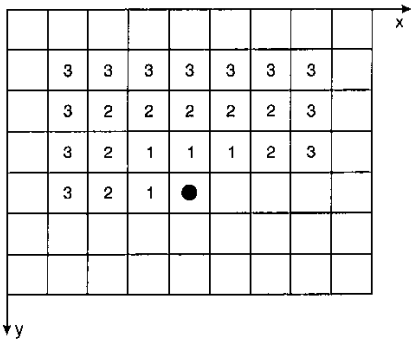

Fig. 1 *Terms used by 2-D causal linear predictor*
If order is K, all terms labelled with number less than or equal to K are used

The order K of the predictor is:

$$K = max(|\alpha_1|, |\beta_1|, \ldots |\alpha_k|, |\beta_k|, \ldots) \tag{2}$$

The pixels are assumed to be ordered by rows. For the 2-D filter to be causal, only the pixels previous to the one being predicted have to be considered. Fig. 1 shows the terms used by a 2-D causal linear predictor of order K; all the terms labelled with a number less than, or equal to, K are used. The number of predictor coefficients is $P = 2K(K+1)$.

The predictor coefficients a_k are chosen such as to minimise the mean square estimation error (MSEE), considering $\boldsymbol{u}[x, y]$ a random variable being estimated in terms of the RVs:

$$\boldsymbol{u}[x - \alpha_k, y - \beta_k] \tag{3}$$

The prediction error $e[x, y]$ is the difference between $u[x, y]$ and its prediction:

$$e[x, y] = u[x, y] - \hat{u}[x, y] \tag{4}$$

The prediction filter (LPF) allows the original image to be reconstructed from its prediction error:

$$u[x, y] = \hat{u}[x, y] + e[x, y] = \sum_k a_k u[x - \alpha_k, y - \beta_k] + e[x, y] \tag{5}$$

The following results have been empirically obtained from a large number of observations:

(i) Predictor coefficients are usually less than 1 in magnitude, but not always. Coefficients greater than 1 in magnitude result in an unstable filter. This can happen because in 2-D the autocorrelation matrix of u is not Toeplitz, a problem that does not arise in 1-D [1].

(ii) The first-order probability density function (PDF) of the prediction error is approximately Gaussian:

$$f(e) \approx \frac{1}{\sigma_e \sqrt{2\pi}} \exp\left(-\frac{e^2}{2\sigma_e^2}\right) \tag{6}$$

(iii) A predictor of order $K = 2$ results in a prediction error with significantly less variance and more decorrelated samples than a predictor of order $K = 1$.

It may be concluded that the stochastic nature of the prediction error is approximately the same for every image to be encoded, which allows to generate and code it by stochastic means without using a training set. It must be kept in mind, however, that our ultimate objective is not the prediction error itself, but the original image.

3 SVQ linear prediction encoder

To encode an image, a linear prediction filter is first computed. Then, the resulting prediction error is encoded using stochastic vector quantisation, meaning that the codebook is created using a random number generator. That is, no training set of images is used to generate the codebook. To decode the image, the prediction error is decoded and then filtered using the prediction filter computed at the encoder. The prediction error is decoded and filtered as a whole to preserve correlation between pixels in adjacent blocks, thus avoiding the block effect found in conventional VQ. The prediction error, however, is encoded without trying to minimise its distortion in any way. It is the distortion of the decoded image at the output of the prediction filter that must be minimised. The prediction error does not need to be computed at all to encode it. The SVQ encoder just selects the codeword that, after filtering at the decoder, minimises the distortion for the original image.

In what follows, the image size will be assumed to be $N \times N$, although the system is well suited for non-square images as well. The system has been designed for grey-scale (or monochrome) images only. To encode a colour image, each of its components (*YUV*) must be encoded separately. The particular steps for implementing the encoder as well as the complexity issues are:

(i) Mean extraction: Compute the mean of the image and remove it.

Complexity: $2N^2$ additions.

(ii) Filter computation: Compute the predictor coefficients that minimise the mean square estimation error for the image. Such coefficients are the solutions of the following system of equations (P is the number of coefficients as described in Section 2):

$$\begin{pmatrix} R[0,0] & R[\alpha_2-\alpha_1, \beta_2-\beta_1] & \ldots & R[\alpha_P-\alpha_1, \beta_P-\beta_1] \\ R[\alpha_2-\alpha_1, \beta_2-\beta_1] & R[0,0] & \ldots & R[\alpha_P-\alpha_2, \beta_P-\beta_2] \\ \vdots & \vdots & \ddots & \vdots \\ R[\alpha_P-\alpha_1, \beta_P-\beta_1] & R[\alpha_P-\alpha_2, \beta_P-\beta_2] & \ldots & R[0,0] \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_P \end{pmatrix} = \begin{pmatrix} R[\alpha_1, \beta_1] \\ R[\alpha_2, \beta_2] \\ \vdots \\ R[\alpha_P, \beta_P] \end{pmatrix} \tag{7}$$

where the correlation function of u is given by:

$$R[\alpha, \beta] = \frac{1}{N^2} \sum_{x,y} u[x, y] u[x - \alpha, y - \beta] \tag{8}$$

Only filters of order 1 and 2 are used (4 and 12 coefficients, respectively). For some images and a given filter order, some of the filter coefficients may be greater than 1 in magnitude and the filter becomes unstable. The problem can be solved changing the order of the filter.

Complexity: $(P(P+1)/2+1)N^2$ multiplications–additions. Also, a system of equations of order P must be solved.

(iii) Computation of the prediction error variance: Compute the variance σ_e^2 of the prediction error.

Complexity: None. It can be done as part of the prediction filter computation and its complexity has already been included there:

$$\sigma_e^2 = E\{e^2\} = R[0,0] - \sum_k a_k R[\alpha_k, \beta_k] \tag{9}$$

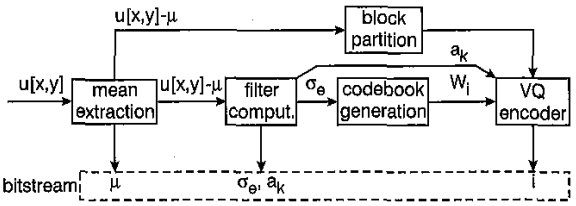

Fig. 2 *SVQLP encoder*

(iv) Codebook generation: Generate a stochastic codebook, $\{W_i\}$, following the prediction error model (a Gaussian PDF). All that is needed for this is to fill the codewords using a Gaussian random number generator. For efficiency reasons (see below), a second codebook $\{V_i\}$ is also generated by feeding the prediction filter with the codewords of the prediction error codebook.

Complexity: LM^2 Gaussian random numbers for the prediction error codebook, $\{W_i\}$, and PLM^2 multiplications–additions for the filtered codebook, $\{V_i\}$, where L is the codebook length (number of codewords) and $M \times M$ is the size of the codewords.

(v) Block coding: Encode the prediction error using stochastic vector quantisation methods (see below).

Complexity: $(3P+L)N^2$ multiplications, $(3P+3L)N^2$ additions.

Fig. 2 shows the SVQLP encoder.

3.1 Stochastic vector quantisation of the prediction error

For every block in the prediction error image, the encoder chooses a codeword from a codebook $\{W_i\}$ that follows its stochastic model. The prediction error itself, however, does not need to be computed because the VQ encoder finds the output that each codeword would produce at the decoder after filtering. It is the distortion of that output that is minimised. Thus, as shown in Fig. 2, the VQ encoder needs to know the codewords for the prediction error, the filter coefficients that will be used by the decoder and the original image u, but not the prediction error itself.

Since the codebook is stochastically generated, the decoder can use exactly the same codewords as the encoder without the need of transmitting them, just using the same seed for the random number generator. Only the codeword indices need to be encoded (together with the mean of the image, the variance of the prediction error and the filter coefficients).

3.2 Prediction error encoding

It is very important to keep in mind two points. First, although we are using SVQ to encode the prediction error, we are not interested in the prediction error itself, but the original image instead. So, it is not important whether the distortion in reproducing the prediction error is minimised or not. The codewords must be chosen to minimise the distortion in reproducing the original image after filtering the prediction error with the prediction filter in the decoder. Secondly, to improve image reproduction, the whole prediction error image is decoded first and then filtered as a whole, rather than filtering each block separately. This means that, when a particular block is encoded, surrounding blocks must be taken into account to evaluate the resulting image after filtering. That is, the best codeword for a particular block depends on which codewords are used to encode surrounding blocks.

The encoder needs to know what the output at the decoder will be if a particular codeword is chosen to encode a given block. The problem is that, due to the autoregressive nature of the prediction filter, given two contiguous blocks to be encoded the best choice for anyone of them depends on what was the choice for the other. That is, assuming that blocks are encoded in sequence (by rows), to choose the best codeword for a particular block we need to know the prediction filter output at pixels located in blocks that have not been encoded yet. The problem arises from the fact that the order in which pixels will be filtered in the decoder (by rows) is not the order in which they are encoded (by blocks), unless the blocks are single-row.

While using single-row blocks could be a solution, it has its drawbacks as well. Hence, a more flexible approach has been devised to allow using blocks of arbitrary dimensions, thus increasing the flexibility of the system. The main objective is to reproduce, as closely as possible, the decoding process at the encoder so that the codewords are chosen taking into account the way they will be used.

3.3 Codebook generation

It is important to realise that we do not need to know exactly the output values produced by encoding a given block with a particular codeword. All we need to know is if that codeword gives better results than any other in the codebook. The influence of a prediction error block in the output image is more relevant for the pixels located in that block. Hence, the pixels located in the block being encoded is where different codewords will produce greater differences in output distortion, so we do not need to evaluate output distortion outside that block. Actually, this is one of the reasons to use square blocks rather than single row ones. If single-row blocks were used, the influence of a block in the blocks below it could not be ignored when encoding it.

In principle, since the LPF is recursive, to compute the output of the prediction filter at a particular pixel, all previous output pixels should have already been computed. The problem is that, as has already been noted, we do not know the values of all such pixels because the blocks they belong to have not been encoded yet. The unknown pixels are those located to the right of the block being encoded. If the reproduction quality is good enough, however, the unknown but needed pixels can be estimated from the original image. That is, we can assume that the pixel values at the decoder output will be 'close' to those of the original image and hence use these instead.

All the output pixels surrounding the block being encoded can thus be estimated, either because their blocks have already been encoded or because we use the original image values instead. This, and the fact that we only evaluate the output distortion at the block being encoded, can dramatically reduce the filtering needed to choose the best codeword for a given block. Whenever a codeword is considered as a candidate for encoding a new block, rather than filtering all the prediction error already encoded (which is what the decoder will do), we consider the output in that block as the result of the superposition of two signals at the input of the prediction filter: (i) the codeword itself (with zero surrounding pixels) and (ii) all the surrounding pixels (with zero values at the location of the block). For a given codeword, the first signal is always the same, regardless of the block being encoded. Thus, we need to filter every codeword W_i only once, before any block is encoded. The result of such pre-filtering is stored

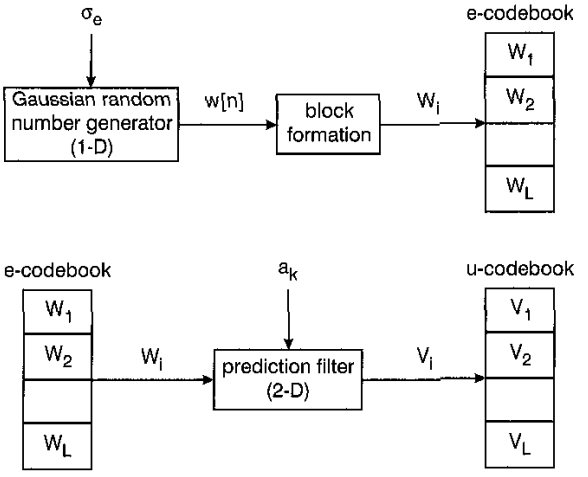

Fig. 3 *Codebook generation*

in what can be viewed as a second codebook, $\{V_i\}$ that reproduces locally the output image statistics.

Fig. 3 shows the two codebooks used by the encoder. Note that the second codebook $\{V_i\}$ is used for efficiency reasons only. It is not used to encode the original image u, even though it locally follows the same statistics. It is used to help the encoder to decide which codewords W_i must be chosen to encode the prediction error (which is itself unknown). In fact, the decoder does not need that second codebook at all. All it needs to know is how to generate the codebook for the prediction error $\{W_i\}$ and the coefficients of the prediction filter.

In summary, for a given block, the encoder chooses the codeword W_i such that V_i plus the effect of filtering an empty (zero) block with the non-zero estimated output surrounding pixels minimises the output *MSEE* for pixels in that block. After encoding a block, it is convenient to re-filter it, together with one or more previous blocks, to avoid error propagation in output estimations for pixels surrounding the next block to be encoded.

3.3 Overall encoder complexity

Given the parameters of the system, the overall encoder complexity is $O(N^2)$. For large codebooks, it can be written as $O(LN^2)$. Thus, the complexity of the encoder increases linearly with the image size and codebook length.

4 SVQ linear prediction decoder

Fig. 4 shows the SVQLP decoder. The particular steps to implement the encoder as well as the complexity issues are:

(i) Codebook generation. Generate the same stochastic codebook for the prediction error $\{W_i\}$ as the encoder. This only requires using the same variance and seed for the random number generator.

Complexity: LM^2 Gaussian random numbers.

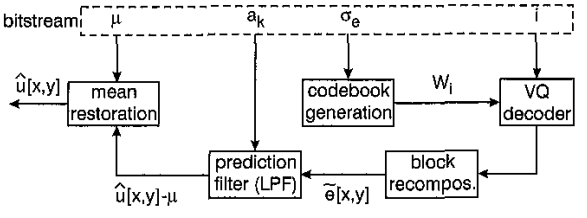

Fig. 4 *SVQLP decoder*

(ii) Block decoding. Decode the prediction error.

Complexity: None (N^2 memory moves).

(iii) Image filtering. Filter the decoded prediction error with the prediction filter.

Complexity: PN^2 multiplications–additions, where P is the number of coefficients of the prediction filter.

(iv) Mean restoration. Restore the mean of the image.*Complexity*: N^2 additions.

4.1 Overall decoder complexity

The overall decoder complexity is $O(N^2)$, regardless of the codebook length. For large codebooks, the encoder complexity can be written as $O(LN^2)$, which means that the decoder is much simpler than the encoder. This property is very important for real time applications.

5 SVQ linear prediction bit stream

To know the details of the SVQ linear prediction implementation, the generated bitstream will be explained now. In addition to the codeword indices, the encoder must send to the decoder the parameters needed to generate exactly the same codebook for the prediction error, the LPF coefficients and the mean of the image. Also, if other parameters such as the image size, filter order, codebook length, codeword size etc. are not known in advance, that information must also be encoded and sent to the decoder. For colour images, each of the components must be encoded separately, but some of those parameters need to be encoded only once. Our current implementation of the system provides bitstreams for both greyscale and color YUV images with chromaticity downsampling (4:2:0).

The first byte of the bitstream is always an arbitrary signed 8-bit code number identifying the format for the rest of the bitstream (i.e. greyscale or YUV 4:2:0).

5.1 Bitstream for greyscale images

- SVQ*Format* (8 bits).
- *ImWidth1* (12 bits): Image width -1. $0 \leq ImWidth1 < 4096$.
- *ImHeight1* (12 bits): Image height -1. $0 \leq ImHeight1 < 4096$.
- *Mean* (8 bits): Mean of the image. $0 \leq Mean < 256$.
- *FiltOrder* (2 bits): Order of the prediction filter. $1 \leq FiltOrder \leq 3$.
- *StdDev* (6 bits): Standard deviation of the prediction error. $0 \leq StdDev < 32$ (uniform quantisation).
- *Coeffs* (8 bits/coeff): Prediction filter coefficients. There are $2^* FiltOrder^* (FiltOrder+1)$ coefficients. $-1 < coeff < 1$ (uniform quantisation).
- *CwdBits* (4 bits): Number of bits per codeword. $0 \leq CwdBits < 16$. The corresponding codebook length is $2^{CwdBits}$, $0 \leq$ codebook length ≤ 32768. If $CwdBits = = 0$, there is no codebook (the prediction error must be generated at random at the decoder) and the *Codewords* field is empty.
- *CwdWidth1* (4 bits): Codeword width -1. Width/height of image blocks. $0 \leq CwdWidth1 < 16$
- *Codewords*: Codeword indices for the image blocks. The number of bits per codeword index is *CwdBits*. The

number of codeword indices is the number of blocks in the image:

ceil(*ImWidth/BlockWidth*)*ceil(*ImHeight/BlockHeight*)

where:

ImWidth = *ImWidth1* + 1,

ImHeight = *ImHeight1* + 1,

BlockHeight = *BlockWidth* = *CwdWidth1* + 1

5.2 Bitstream for colour YUV 4:2:0 images

- SVQ*Format* (8 bits).
- *ImWidth1* (12 bits): image width − 1.
- *ImHeight1* (12 bits): image height − 1.
- *Mean_Y* (8 bits): mean of the Y component.
- *Mean_U* (8 bits): mean of the U component.
- *Mean_V* (8 bits): mean of the V component.
- *FiltOrder_Y* (2 bits): order of the prediction filter for Y.
- *StdDev_Y* (6 bits): standard deviation of the prediction error for Y.
- *FiltOrder_U* (2 bits): order of the prediction filter for U.
- *StdDev_U* (6 bits): standard deviation of the prediction error for U.
- *FiltOrder_V* (2 bits): order of the prediction filter for V.
- *StdDev_V* (6 bits): standard deviation of the prediction error for V.
- *Coeffs_Y* (8 bits/coeff): prediction filter coefficients for Y.
- *Coeffs_U* (8 bits/coeff): prediction filter coefficients for U.
- *Coeffs_V* (8 bits/coeff): prediction filter coefficients for V.
- *CwdBits_Y* (4 bits): number of bits per codeword for Y. If *CwdBits_Y* = = 0, there is no codebook for Y and the *Codewords_Y* field is empty.
- *CwdWidth1_Y* (4 bits): codeword width − 1 for Y.
- *CwdBits_UV* (4 bits): number of bits per codeword for both U and V. If *CwdBits_UV* = = 0 there is no codebook either U nor V and the fields *Codewords_U* and *Codewords_V* are empty.
- *CwdWidth1_UV* (4 bits): Codeword width-1 for both U and V
- *Codewords_Y*: Codeword indices for the Y component. The number of bits per codeword index is *CwdBits_Y*. The number of codeword indices is the same as for greyscale images.
- *Codewords_U*: Codeword indices for the U component. The number of bits per codeword index is *CwdBits_UV*. The number of codeword indices is the same as for a greyscale image having half the number of rows and columns of the Y component.
- *Codewords_V*: Codeword indices for the V component. The number of bits per codeword index is *CwdBits_UV*. The number of codeword indices is the same as for a greyscale image having half the number of rows and columns of the Y component.

6 Region-based SVQ linear prediction

The SVQ linear prediction scheme has been explained so far having in mind rectangular images. However, it is increasingly important to develop schemes that take into account the shape of the object to be encoded. As an example, let us mention that the new MPEG-4 standard may represent and manipulate objects of arbitrary shape. It is in this context that we have also coped with this situation and the SVQ linear prediction scheme has been adapted to encode textures of any shape.

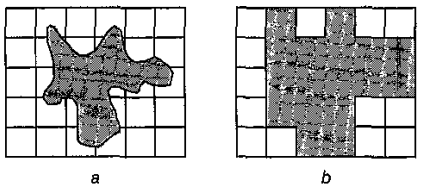

Fig. 5 *Encoding of arbitrary image*

a Block partition of region

b Modification of region limits to encode/decode prediction error

The scheme just described is applicable to textures only. That means that the image must be homogeneous in some sense. This is because contours of an arbitrary image cannot be preserved after filtering by the LPF, and also because a single mean and variance are computed for the whole image. The proposed scheme can be used, however, to encode arbitrary images which have been segmented first. The system can then be used to encode each of the (more or less) homogeneous regions comprising the image. Thus, the system fits well in object-based image coding systems.

As with non-segmented images, the encoder must still be able to reproduce the decoding process to decide which codeword is assigned to a particular block of the prediction error. Since the codewords V_i used by the encoder are obtained filtering W_i completely, in order to apply superposition the prediction error must be coded as if the region occupied completely any block in which it contains at least one pixel (Fig 5). Of course, distortion must be evaluated only within the true region limits and the decoded region must recover its original form after the prediction error has been filtered through the LPF.

To avoid wasting bits when encoding blocks with only a small number of pixels in the region, those blocks can be coded with a randomly chosen codeword; encoder and decoder must choose the same 'random' codeword because the encoder must take into account their effect on the surrounding blocks after filtering at the decoder.

7 Results

The SVQ scheme has been applied to a variety of texture images with similar results. We have selected here the 512 × 512 colour images 'Brick' and 'Fabric' from the MIT data base used in core experiments X1 and Z1 of MPEG-4 SNHC [8]. As the SVQ linear prediction approach is intended for very high compressions, only results in the range of compression ratios between 64 and 256 will be shown. Visual results below these compression factors are practically identical to DCT techniques. In addition, let us mention that, although the peak signal to noise ratio (PSNR) measure has been used extensively in the evaluation of image coding performance, in the case of high compression ratios it does not provide a reliable measure of performance as it does not match the visual criterion of quality. For this reason, only visual results will be provided.

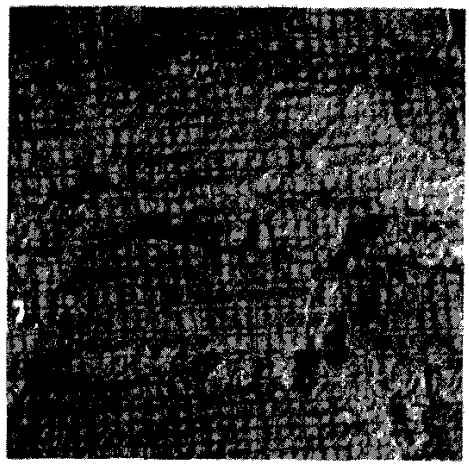

Fig. 6 *Original image 'Brick'*

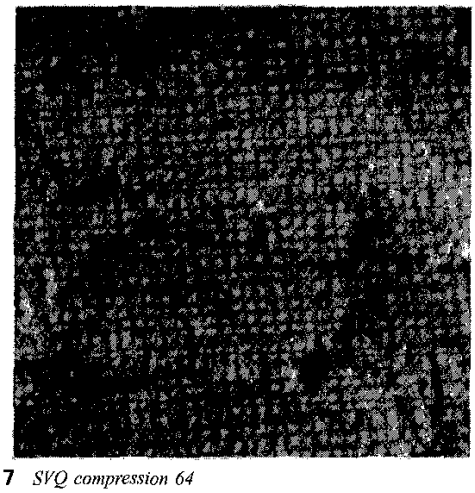

Fig. 7 *SVQ compression 64*

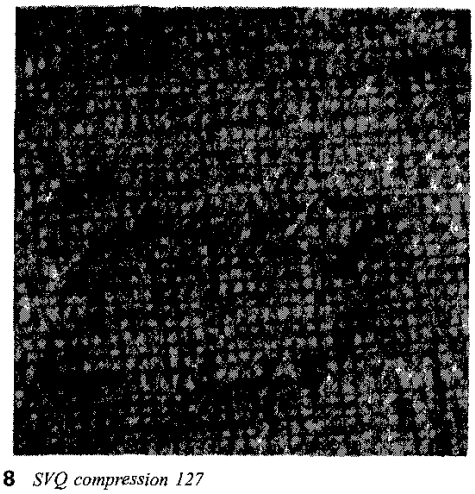

Fig. 8 *SVQ compression 127*

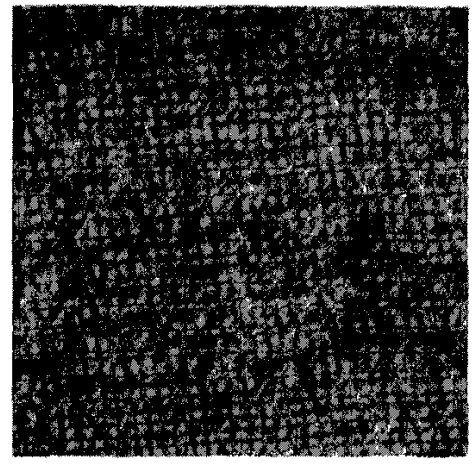

Fig. 9 *SVQ compression 259*

Fig. 10 *VM compression 64*

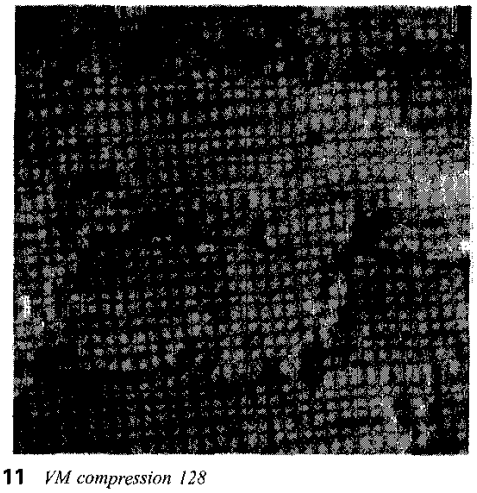

Fig. 11 *VM compression 128*

Figs 6, 7, 8 and 9 show the original 512×512 image 'Brick' and the results of the SVQ linear prediction scheme for compression factors of 64, 127 and 259, respectively. Correspondingly, Figs 10, 11 and 12 show the results of the verification model (VM) of MPEG-4 when applied to the Brick image for compression factors of 64, 128 and 256 respectively. This model is DCT based. Please notice that only the left upper 256×256 portion of the images is shown.

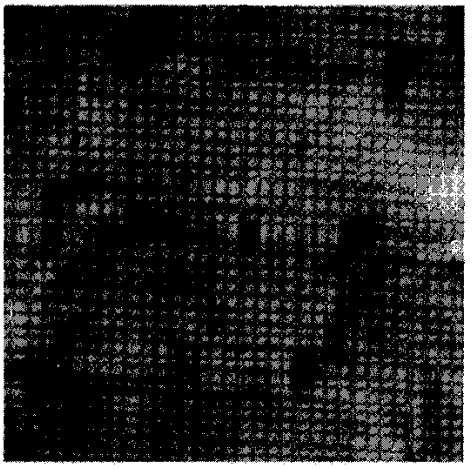

Fig. 12 *VM compression 256*

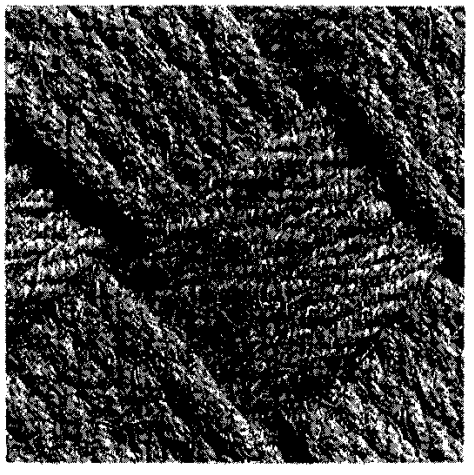

Fig. 13 *Original image 'Fabric'*

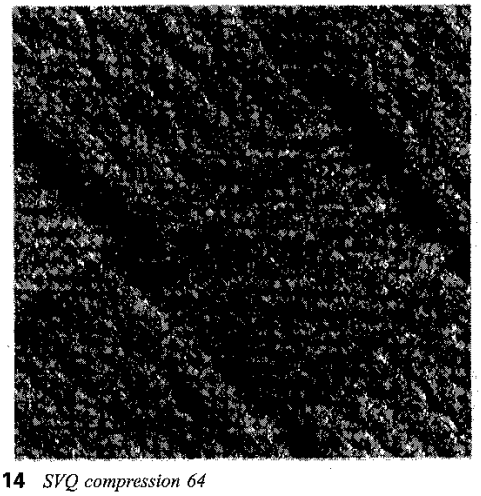

Fig. 14 *SVQ compression 64*

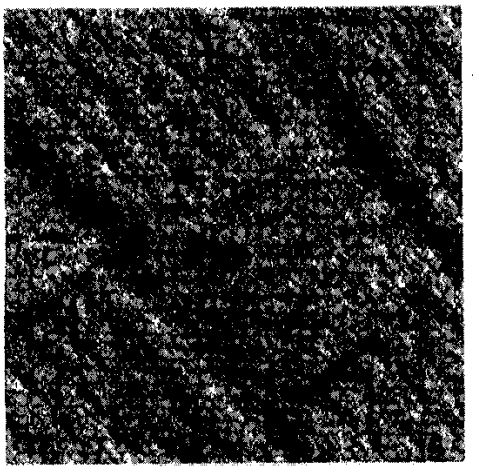

Fig. 15 *SVQ compression 127*

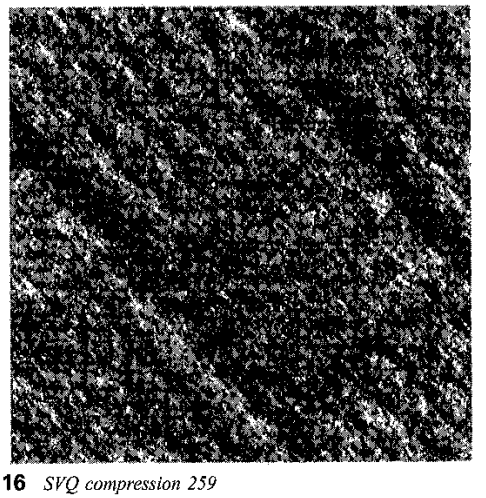

Fig. 16 *SVQ compression 259*

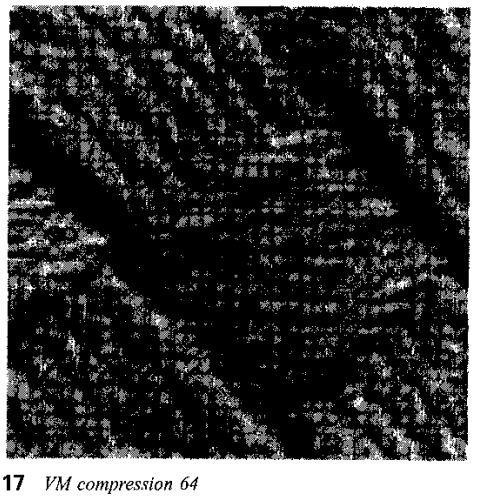

Fig. 17 *VM compression 64*

Figs. 13, 14, 15 and 16 show the original 512×512 image 'Fabric' and the results of the SVQ linear prediction scheme for compression factors of 64, 127 and 259, respectively. Correspondingly, Figs. 17, 18 and 19 show the results of the VM of MPEG-4 when applied to the 'Fabric' image for compression factors of 64, 128 and 256, respectively. Please notice that only the left upper 256×256 portion of the images is shown.

In the case of homogeneous textures and low compression, SVQ linear prediction provides equal visual quality than the video VM. In the case of homogeneous textures and high compression (bigger than 64), SVQ linear predic-

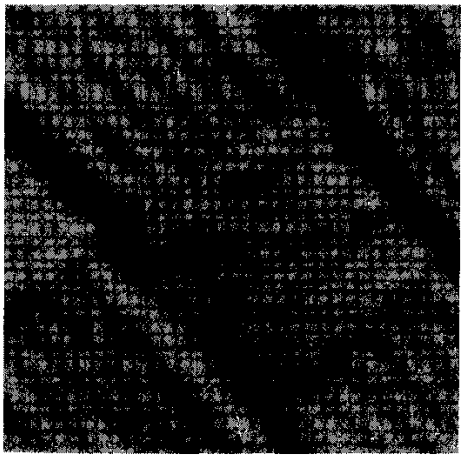

Fig. 18 *VM compression 128*

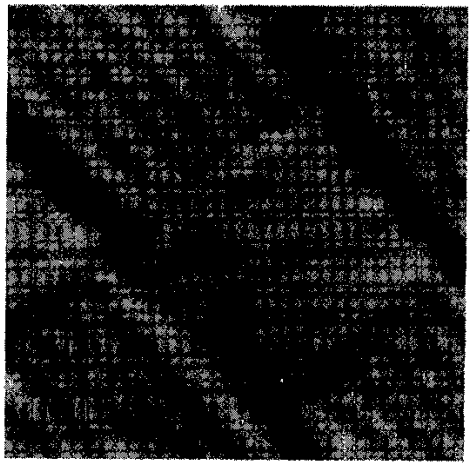

Fig. 19 *VM compression 256*

tion provides always better appealing quality than current video VM.

Generally speaking, it can be stated that, for homogeneous textures, the video VM has to work with images downsampled by a factor of 1 when compression ratios of or around 32 have to be achieved. The downsampling factor has to be increased till 2 if compression factors of or bigger than 64 have to be achieved. The downsampling and corresponding upsampling result in a blurred reconstructed image. The SVQ linear prediction approach does not need downsampled images to achieve compression factors bigger than 32. For compression ratios bigger than 64, the results show that SVQ linear prediction provides better visual results than the MPEG-4 VM. In addition, comparisons made in the context of MPEG-4 core experiments, also show that SVQ linear prediction provides better visual results than the wavelet approach proposed in the SNHC group of MPEG-4 [8]. SVQ can extend the compression factors beyond 256 with acceptable quality depending on the application. It is also important to note that, at this time, no effort has been made to use an entropic coder for the SVQ linear prediction technique. It can be expected that an increase of about 10–15% in compression ratio can be achieved if entropic coders are used.

8 Conclusions

A new technique for efficient coding of homogeneous textures has been presented. The approach relies on stochastic vector quantisation linear prediction and provides compression ratios as high as 64, 128 and 256. When compared against the current texture coding proposals of the MPEG-4 standard, it provides better visual quality for those compression ratios. Complexity issues are very acceptable and the decoder can be implemented very efficiently as in conventional VQ schemes.

9 Acknowledgments

This work has been partially supported by the VIDAS ACTS project of the European Union and by TIC 95-1022-C05-05 of the Spanish Government

10 References

1 GRAY, R.M.: 'Vector quantization', *IEEE ASSP Mag.*, 1984, **1**, (2), pp. 4–29
2 GERSHO, A., and GRAY, R.M.: 'Vector quantization and signal compression' (Kluwer Academic Publishers, 1992) pp. 626–628
3 LINDE, Y., BUZO, A., and GRAY, R.M.: 'An algorithm for vector quantizer design', *IEEE Trans.*, 1980, **28**, pp. 84–95
4 TORRES, L., CASAS, J. R., and ARIAS, E.: 'Stochastic vector quantization of images', *Signal Process.*, 1997, **62**, (3), pp. 291–301
5 GIMENO, D., TORRES, L., and CASAS, J.R.: 'A new approach to texture coding using stochastic vector quantization', IEEE international conference on *Image processing*, ICIP, 1994, Austin, USA , pp. 119–123
6 GUPTA, S., and GERSHO, A.: 'Image vector quantization with block adaptive scalar prediction', *Proc. SPIE -Int Soc. Opt. Eng.*, 1991, **1605**, pp. 179–189
7 NGUYEN, H.Q., and DUBOIS, E.: 'A 2-dimensional code excited linear prediction (2D-CELP) image coding system', SPIE/SPSE symposium on *Electronic imaging*, 1990, Santa Clara, Ca, pp. 190–198
8 'Results of core experiments X1 and Z1'. MPEG Documents M1706–1900 ISO/IEC JTC1/SC29/WG11, 1997